Finding dense locations in indoor tracking data

Ahmed, Tanvir; Pedersen, Torben Bach; Lu, Hua

Published in:
Proceedings of the 15th IEEE International Conference on Mobile Data Management (MDM)

DOI (link to publication from Publisher):
10.1109/MDM.2014.29

Publication date:
2014

Document Version
Accepted author manuscript, peer reviewed version

Finding Dense Locations in Indoor Tracking Data

Tanvir Ahmed Torben Bach Pedersen Hua Lu
Department of Computer Science, Aalborg University, Denmark
Email: {tanvir, tbp, luhua}@cs.aau.dk

***Abstract*—Finding the dense locations in large indoor spaces is very useful for getting overloaded locations, security, crowd management, indoor navigation, and guidance. Indoor tracking data can be very large and are not readily available for finding dense locations. This paper presents a graph-based model for semi-constrained indoor movement, and then uses this to map raw tracking records into mapping records representing object entry and exit times in particular locations. Then, an efficient indexing structure, the Dense Location Time Index (DLT-Index) is proposed for indexing the time intervals of the mapping table, along with associated construction, query processing, and pruning techniques. The DLT-Index supports very efficient aggregate point queries, interval queries, and dense location queries. A comprehensive experimental study with real data shows that the proposed techniques can efficiently find dense locations in large amounts of indoor tracking data.**

I. Introduction

Technologies like RFID and Bluetooth enable a variety of indoor tracking applications like tracking people's movement in large indoor spaces (e.g., airport, office building, shopping mall, and museums), airport baggage tracking, items movement tracking in supply chain system, etc. The huge amount of tracking data generated by these types of systems is very useful for analyses and decision making. These analyses are useful for different kinds of location-based services, finding problems in the systems, and further improvement in the systems. Unlike GPS based positioning for outdoor systems, indoor tracking provides the symbolic locations of the objects in indoor space. Examples of symbolic locations include security and shopping areas in airports, and the different sections of rooms in museum exhibitions. In airports, the bags pass different symbolic locations in each step like check-in, screening, sorting, etc. Finding the dense or overloaded baggage handling locations helps handling bags more efficiently. For passengers moving in an airport, detecting dense locations shows where and when passengers gather, and can be used for crowd management and providing location-based passenger services.

In our previous short paper [1], we proposed a graph based model and mapping technique for capturing dense locations in constrained indoor movement only. In the present paper, we additionally model semi-constrained indoor movement and provide technique that maps the indoor tracking records into mapping records with the entry/exit times of an object at a symbolic location. The derived mapping records can be used for finding dense locations as well as for other analyses like stay duration estimation, etc. We also propose an efficient indexing structure, the Dense Location Time Index (DLT-Index), which stores aggregate information like the number of objects entering, exiting, and present at a location at different timestamps or time intervals. Additionally, we provide efficient techniques for index construction and processing dense location queries (as well as point and interval queries) on large data sets, and an efficient pruning technique for the DLT-Index. Finally, we perform a comprehensive experimental evaluation with real data showing that the proposed solutions are efficient and scalable.

The remainder of the paper is organized as follows. Section 2 gives the problem formulation. Section 3 describes the graph-based models and the mapping of tracking records. Section 4 presents the DLT-Index and the associated query processing and pruning techniques. Section 5 presents the experimental evaluation. Section 6 reviews related work. Finally, Section 7 concludes the paper.

II. Problem Formulation

The movements of objects inside indoor space varies with the structure of the paths. Based on the path structure we categorize the indoor spaces into two categories.

Constrained Path Space(CPS): In a constrained path space (CPS), objects move continuously from one symbolic location to another. The objects cannot move freely inside the locations and the locations are in some sense one dimensional. The size of locations inside CPS is measured by length not by area.

Example of a CPS can be the conveyor belt system of an airport baggage handling system. The conveyor is divided into different symbolic locations like check-in belts, screening belts, sorter belts, etc. More detail about the constrained indoor movement can be found in [1] and about the baggage tracking process in [2].

Semi-Constrained Path Space (SCPS): In a semi-constrained path space (SCPS), the objects move more freely compared to a CPS. The objects move from one symbolic location to another and they can also stay some period of time inside the locations. The locations are two-dimensional. The size of SCPS locations is measured by area not by length.

Examples can be movement of people between rooms in office space or museums, different sections in airports etc. Fig. 1 shows an example of an SCPS. The indoor space is divided into different symbolic locations e.g., rooms, hall ways etc. For entering and exiting a room, an object has to cross the entry/exit points (e.g., doors). Some entry/exit points are unidirectional and some are bi-directional. The arrows represent the unidirectional movements.

In our setting, the tracking devices are strategically deployed at different fixed locations inside the indoor space, e.g., each

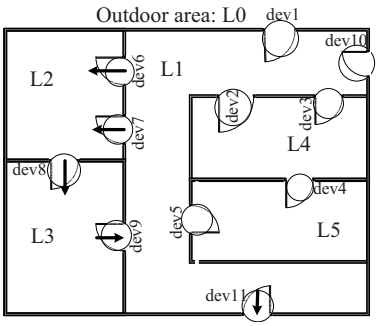


Fig. 1: Semi-constrained space

section of conveyor belts, door of a room, between sections of a hallway etc. The objects contain tags (e.g., RFID tags, Bluetooth devices, etc.,) that can be tracked by the tracking devices (e.g., RFID readers, Bluetooth access points etc.,). In Fig. 1 the circles represent the deployment of the RFID readers and their tracking ranges. When an object comes under a tracking device's activation range, it is continuously detected by the tracking device with a sampling rate and it generates raw reading records with the form: *(trackingDeviceID, ObjID, t)*. It means that a tracking device *trackingDeviceID* detects a moving object *ObjID* in its activation range at timestamp *t*. A *TrackingRecord(recordID, ObjID, TrackingDeviceID, time$_{in}$, time$_{out}$)* table is constructed from the raw tracking sequence, where *recordID* is record identifier and *time$_{in}$, time$_{out}$* respectively represent the timestamps of first reading and last reading of *ObjID* by *TrackingDeviceID* in its activation range. An example table for tracking records of an object o_1 at floor plan of Fig. 1 is shown in Table I. Here, the record *rec3* means that o_1 is observed by device *dev3* from time 15 to 18.

TABLE I: Tracking Records of Indoor Moving Objects

RecordID	ObjID	TrackingDeviceID	time$_{in}$	time$_{out}$
rec1	o_1	*dev1*	*4*	*5*
rec3	o_1	*dev3*	*15*	*18*
rec5	o_1	*dev4*	*26*	*29*
rec8	o_1	*dev4*	*51*	*54*

Problem Definition. Let L be the set of all symbolic locations inside a large indoor space, $L = \{l_1, l_2, l_3, \ldots, l_k\}$. The *capacity* of location l_i is denoted by c_i = *capacity*(l_i).

Definition 1 (Capacity). The *capacity* of a location l_i is the numbers of objects that can appear in l_i during a given time unit.

For example, the *capacity* of room $L3$ of Fig. 1 can be 20 persons per 15 min.

Definition 2 (Density). Let n_i be the number of objects appearing in location l_i during the time interval, $w = [t_{start}, t_{end}]$ and c_i = capacity(l_i) be the capacity of location l_i. Then the *density* of location l_i for interval w is defined as,

$d_i = \frac{n_i}{\Delta t \times capacity(l_i)} \times 100\%$, where $\Delta t = t_{end} - t_{start}$.

Thus, the *density* shows how dense a location is, as a percentage value.

Definition 3 (Dense Location). A location l_i can be considered as a *dense location (DL)* for interval w if d_i exceeds a given threshold θ.

Definition 4 (Dense Location Query). A *dense location query (DLQ)* finds all the *dense locations* $\subseteq L$, for time interval w.

III. Semantic Location Mappings

A location is typically not fully covered by a tracking device. Moreover, a tracking record contains only the first and last times an object appeared inside the activation range of a tracking device. As a result it is not directly available when an object actually entered (*time$_{start}$*) and exited (*time$_{end}$*) the corresponding location. So mapping strategies are required for retrieving such location and timing information.

As mentioned earlier, in our previous short paper [1] we modeled constrained indoor movement like CPS and described how to convert tracking records into mapping records that contain the entry time (*time$_{start}$*) and exit time (*time$_{end}$*) of objects at different locations. For CPS, some locations contain only one tracking device deployed at any point in each of them and some adjacent locations may not contain tracking devices. An extended reader deployment graph (ERDG) was proposed that contains various topological information like entry lag distance(ENLD), exit lag distance(EXLD), covered distance(CD) etc. The timing information from the tracking records and the corresponding *CD* from the ERDG is used to calculate the speed of the object at the corresponding location. Depending on the topological structure, the nodes were categorized into 5 types. All this information helped for finding the entry and exit time of an object at a CPS location. All details can be found in [1].

In an SCPS, a tracking device is deployed at each entry and exit points of a location. Moreover, the movement can be both uni-directional and bi-directional. For example in Fig. 1 the door containing *dev1* can be used for both entering and exiting the location *L1* and the door with *dev6* can be used only to enter *L2*. Since in SCPS the tracking devices are deployed in the entry and exit points of a location, it is easier to find the entry and exit times of objects for this type of locations from the tracking records compared to CPS. Thus, calculating the speed of the objects is not needed and thus the parameters *ENLD, EXLD* and *CD* are not useful for SCPS. However, we also need to consider that there can be multiple entry and exit points in an SCPS location. A Reader Deployment Graph (RDG) [5] is used for modeling SCPS. Fig. 2 shows the *RDG* for the floor plan given in Fig. 1. The *RDG* is a directed graph where each symbolic location is represented as node and the connection between the locations are represented as edge. Each edge is labeled by the tracking devices deployed between the locations. For mapping, we define a mapping function called *Dest:* $\{l, d\} \rightarrow l'$, where $l, l' \in L$ and $d \in D$. The function*Dest(l, d)* returns node l' from the graph where d is the label for the edge *E* $(l, l') \in E$. It means that, an object traveling from l to l' should be detected by d. For mapping, initially the location of an object o is assumed to be in the outdoor location *L0*. Then each tracking record of o is accessed and it is determined where o is entering with the help of *Dest* function. Then the *time$_{in}$* in the tracking record becomes *time$_{start}$* and the *time$_{in}$*

of the next tracking record becomes *time*$_{end}$ for the entered location. For example in Table I, record *rec1* represents that o_1 was tracked by *dev1* from time 4 to 5. The initial location of o_1 is considered as $L0$. From the graph in Fig. 2, *Dest(L0, dev1) = L1*. From tracking record, *time*$_{start}$ = *time*$_{in}$ = 4. However, the *time*$_{end}$ = *time*$_{in}$ *from rec3* = 15. So the mapping record says that o_1 was at $L1$ from time 4 to 15. Table II shows the SCPS mapping results for Table I.

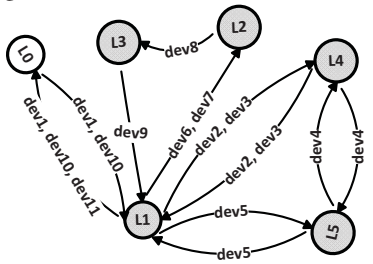


Fig. 2: Graph based model for SCPS

TABLE II: MappingTable for Table I considering SCPS

MappingID	ObjectID	LocationID	time$_{start}$	time$_{end}$
map1	*o1*	*L1*	*4*	*15*
map3	*o1*	*L4*	*15*	*26*
map5	*o1*	*L5*	*26*	*51*
map5	*o1*	*L4*	*51*	...

IV. Efficient Dense Location Extraction

The dense locations can now be extracted from the mapping records in *MappingTable*. As seen, a mapping record contains the entire stay of an object inside a location. For a CPS location like baggage on a conveyor belt, it is not common an object appears multiple times at same location whereas it is common for an SCPS location. In our setting, an object visiting a location multiple times is treated as multiple objects for density computation for that location, since a reappearance contributes to the density.

A DLQ has to access aggregate information for a given time interval from a large amount of data from the mapping table. We develop an indexing technique for that, where the temporal indexing part is motivated by the *time index* described in [3]. Instead of indexing all the records of the mapping table we index all the intervals of each location using separate trees and store aggregate values instead of pointers to the leaf nodes. Moreover, unlike the B^+-tree we link the nodes of intermediate levels. All these improvements let us avoid accessing detailed data records and further offers significant pruning opportunities. This new index structure is called the *Dense Location Time Index* (DLT-index).

A. The DLT-Index.

In the DLT-Index for each location, we maintain a separate tree, called DLT-tree. Let us consider a set of mapping records at location $L1$ shown in Fig. 3a. Fig. 3b shows the DLT-Index constructed for the data given in Fig. 3a. In our DLT-tree, each leaf node entry at time point t_i is of the form: $\langle t_i, C_i \rangle$, where $C_i \langle c_{total}, c_{enter}, c_{exit} \rangle$ is a tuple with some aggregate information of the objects valid during [t_i,t_i^+) where t_i^+ is the next indexed time point, c_{total} and c_{enter} are the total number of objects available and entered at t_i respectively and c_{exit} = total number of objects that exited at t_i-1. Besides, each non-leaf entry at time point t_i contains a tuple $C_{nl}(t_i)$ which is of the form:$\langle c'_{total}, c'_{enter} \rangle$ and their values can be described as:

1) For the left-most entry of a level: c'_{total} = c'_{enter} = total number of objects entered or available until t_i-1.
2) For the entries other than left-most entry: c'_{total} and c'_{enter} are the total number of objects available and entered during interval [t_i^-, t_i) respectively, where t_i^- is the immediate left entry of t_i in the same level.

In addition to C_{nl}, the right-most entry t_i of each non-leaf level also contains another tuple $C_r \langle c''_{total}, c''_{enter} \rangle$ where c''_{total} = total number of objects available from t_i to max time stamp in the tree and c''_{enter} = total number of objects entered from t_i to the max time stamp in the tree.

Tree construction and insertion of a new entry in the DLT-tree is very similar to the B^+-tree. The time points are keys and aggregate information C are the data values. The value of C for the leaf levels can be precomputed or can be computed while inserting. After constructing the tree, the aggregate information of the non-leaf entries and the links between the non-leaf nodes have to be established.

B. Tree Construction from Historical Data

During the tree construction the historical data are indexed. Let the set of all intervals available in the data set be $I = \{I_1, I_2, ..., I_n\}$. For an interval I_i, the value of $I_i.t_s$ and $I_i.t_e$ represents the start and end time respectively. Additionally the value of $I_i^\dashv = I_i.t_e + 1$ represents the next timestamp after $I_i.t_e$. The DLT-Index has to index all the time points of P where P can be defined as follows:

$$P = \{t_i | \exists I_j \in I((t_i = I_j.t_s) \vee (t_i = I_j^\dashv))\}$$

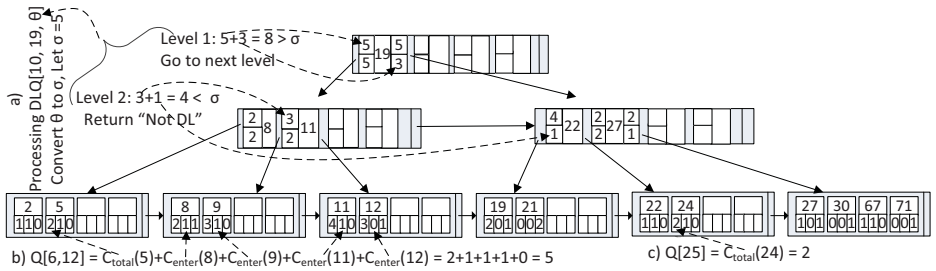
For example, considering the table in Fig. 3a, the points that need to be indexed are P = {*2, 5, 8, 9, 11, 12, 19, 21, 22, 24, 27, 30, 67, 71*}. As seen for *MapID* $r1$, the $time_{end}$ =7 is not included in P as we index the next timestamp of 7 which is 8. Also at time point 8, o_{18} has entered the location (*MapID* $r3$). Before the tree construction all the time points of P are sorted in ascending order into P_s. Each of the time point $t_i \in P_s$ additionally contain a C_i where c_{enter} and c_{exit} are directly known while getting each element of P_s from the data set. For example, at the time point 8: c_{enter} = 1 and c_{exit} = 1, at time point 21: c_{enter}=0 and c_{exit} = 2 as two objects has $time_{end}$ at 21-1 = 20. The c_{total} at time point t_i is calculated by Eq. (1). For example, at initial stage C_0 = $\langle 0, 0, 0 \rangle$. For the first time point 2, c_{enter}=1 and c_{exit} = 0. So, c_{total} = 0+1-0=1. As a result C_1 = $\langle$1,1,0$\rangle$. Similarly for time point 5 c_{total} = 1+1-0=2 and C_2 = $\langle$2,1,0$\rangle$. The complete list of C_i can be found in the leaf nodes of Fig. 3b.

$$c_{total}(t_i) := c_{total}(t_{i-1}) + c_{enter}(t_i) - c_{exit}(t_i) \quad (1)$$

The insertion of the time points are now same as B^+-tree where the time points are keys and C are the data values. In

MapID	ObjID	LocID	time_{start}	time_{end}
r1	o_1	*L1*	*2*	*7*
r2	o_2	*L1*	*5*	*11*
r3	o_{18}	*L1*	*8*	*20*
r4	o_{15}	*L1*	*9*	*18*
r5	o_{16}	*L1*	*11*	*20*
r6	o_{12}	*L1*	*67*	*70*
r7	o_{19}	*L1*	*22*	*26*
r8	o_{20}	*L1*	*24*	*29*

(a) Example mapping records of location $L1$



(b) DLT-Index

Fig. 3: Example mapping records and DLT-Index

addition to the insertion, the nodes at the non-leaf levels have to be linked like *level 2* of Fig. 3b. Moreover, for the entries of the non-leaf levels, the values of C_{nl} and C_r have to be calculated. Maintaining the aggregate information in the leaf nodes gives advantage for such calculation. For any non-leaf entry t_i, if it is the left-most entry in its level then the C_{nl} can be calculated as follows:

$C_{nl}(t_i).c'_{total} = C_{nl}(t_i).c'_{enter} = c_{total}(t_1) + \sum_{j=2}^{i-1} c_{enter}(t_j)$

Besides, for any non-leaf entry t_i, if it is not the left-most entry in its level then the C_{nl} can be calculated as follows:

Let t_i^- be the entry immediately before t_i in its level and k be the position of t_i^- in the leaf node entries. Then,

$$C_{nl}(t_i).c'_{total} = c_{total}(t_k) + \sum_{j=k+1}^{i-1} c_{enter}(t_j)$$
$$C_{nl}(t_i).c'_{enter} = \sum_{j=k}^{i-1} c_{enter}(t_j)$$

Similarly for the right-most entry t_i of a non-leaf level, C_r can be calculated as follows:

$$C_r(t_i).c''_{total} = c_{total}(t_i) + \sum_{j=i+1}^{max} c_{enter}(t_j)$$
$$C_r(t_i).c''_{enter} = \sum_{j=i}^{max} c_{enter}(t_j)$$

where t_{max} represents the maximum value of the indexed time point in the leaf level.

For example in Fig. 3b at level 2, 8 is the left-most entry. So $C_{nl}(8).c'_{total} = C_{nl}(8).c'_{enter} = c_{total}(2) + c_{enter}(5)$ = 1+1 = 2. Considering 22 of level 2 which is not the left-most entry, $C_{nl}(22).c'_{total} = c_{total}(11) + c_{enter}(12) + c_{enter}(19) + c_{enter}(21)$ = 4+0+0+0 = 4. In the example tree 27 is the right-most entry of level 2. The value of $C_r(27).c''_{total} = c_{total}(27) + c_{enter}(30) + c_{enter}(67) + c_{enter}(71)$ = 1+0+1+0 = 2.

C. Query Processing

Here we will discuss how an aggregate point and interval query can be executed from our DLT-Index and then show the dense location query processing with and without pruning.

Aggregate query: There are two types of aggregate queries: a) point queries b) interval queries. A point query finds the total number of objects at a particular time *point*. An interval query finds the total number of objects in a particular time *interval*. For processing a point query $Q[q_t]$, a B^+-tree search is performed for finding the appropriate leaf node for q_t. Then it finds the last entry $q_a \leq q_t$ in the node and returns $C_{total}(q_a)$ as the result. For example consider a point query $Q[25]$. The B^+-tree search will find the node shown in Fig 3bc. From the resulting node, entry 24 is the last entry which is ≤ 25. So the query will return $c_{total}(24) = 2$. Conversely for an interval query $Q[q_s, q_e]$, it will first process a point query $Q[q_s]$ and take $C_{total}(q_a)$. However, instead of returning the result it continues further processing and finds the last entry $q_b \leq q_e$ from the leaf nodes. For this purpose it may has to access consecutive leaf nodes one after another. If $q_a = q_b$ then it returns $C_{total}(q_a)$ as result. Otherwise it adds all the C_{enter} for each entry after q_a until q_b. For example consider an interval query $Q[6, 12]$. The B^+-tree search will find the leaf node containing $\{2, 5\}$ as shown in Fig. 3bb. Then it finds the entry 12 which is the last entry ≤ 12. So the calculation of the result will be: $Q[6, 12] = c_{total}(5) + c_{enter}(8) + c_{enter}(9) + c_{enter}(11) + c_{enter}(12)$ = 2+1+1+1+0 = 5.

Dense Location Query: The naive approach of processing a $DLQ[q_s, q_e, \theta]$ over the DLT-Index is to just get the results of an interval query $Q[q_s, q_e]$ for each of the locations, compute the density from the result, and then determine which locations are DLs. However, for each of the cases, the query has to access the leaf nodes and then compute the aggregation. The access of the nodes of the leaf level as well as other level can be reduced by our pruning technique. As described the DLT-Index contains some aggregate information in the non-leaf node entries for pruning purpose. We first introduce the following two observations.

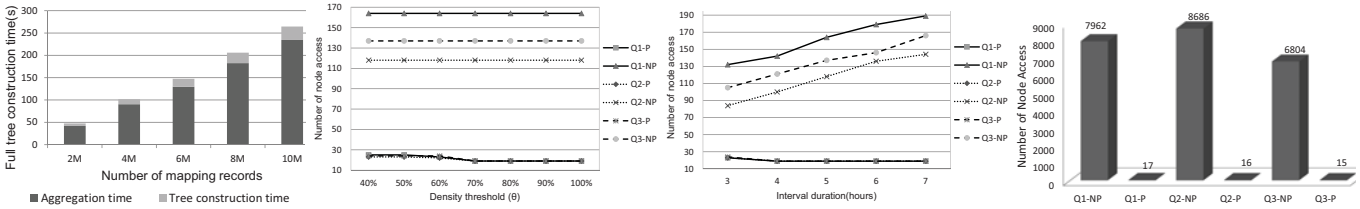
Observation 1 (obs 1): If the number of objects in an interval i_1 is less than a threshold k, a smaller interval i_2 that is fully covered by i_1 must have less than k objects.

Observation 2 (obs 2): If the number of objects in an interval i_1 is greater than a threshold k, a larger interval i_2 that fully covers i_1 must have more than k objects.

In our pruning technique the first observation helps to prune out the non-leaf levels, whereas the second observation helps to prune at the leaf level. Note that the cases covered by obs 1 and 2 are mutually exclusive. In a $DLQ[q_s, q_e, \theta]$ the given θ is a density threshold, not a number of objects. Before starting the query processing we convert θ to a number of object threshold σ_i for each location l_i. The value of σ for a location L_i for a $DLQ[q_s, q_e, \theta]$ can be derived from the density formula which is given below:

$$\sigma(L_i, \theta) = \frac{\Delta t \times capacity(L_i) \times \theta}{100}, \text{ where } \Delta t = q_e - q_s.$$

While processing a $DLQ[q_s, q_e, \theta]$, the value of σ for each location has to be calculated. The query has to traverse the tree of each location. The way of accessing the next level is similar to B^+-tree search for q_s. At each non-leaf level l_i of the DLT-tree of a location Loc_i, the smallest interval $[t_a, t_b)$ that can cover $[q_s, q_e]$ has to be found. After that, the total



(a) Tree construction time (b) Effect of θ in DLQ processing (c) Effect of Δt in DLQ processing (d) Interval group query

Fig. 4: Experimenting with different aspects of the DLT-Index.

number of objects $n(l_i)$ during $[t_a, t_b)$ is calculated from the C_{nl} stored in all the entries between t_a and t_b in level l_i. The value of $n(l_i)$ is compared with σ. If $n(l_i) < \sigma$ then Loc_i is marked as 'Not DL' based on *obs 1* and further processing of the tree is pruned. However, if $n(l_i) \not< \sigma$ then the next level is accessed and continue further processing in the similar way. If the query reaches the leaf level then it starts calculating the total number of objects during $[q_s, q_e]$. During the calculation, at each step it compares the sum with σ and if $sum > \sigma$ then based on 'emphobs 2 the location Loc_i is marked as DL. As a result it skips the access of the next entries and nodes and avoids further calculation. If the condition $sum \leq \sigma$ then the full sum is calculated to decide whether Loc_i is DL or not.

Consider Fig. 3ba, where the processing of *DLQ[10, 19, θ]* is shown for location L_1. Let $\sigma = 5$ be derived from θ. At the root level, the smallest interval that covers $[10, 19]$ is [*min, max*) where *min* and *max* are the starting and ending time points of the tree respectively. For min the $C_{nl}.c'_{total}$ from the left-most entry is taken, and for max $C_r.c''_{enter}$ is taken from the right-most entry of the current level. So in our case the total number of objects during this period is $C_{nl}(19).c'_{total}$ + $C_r(19).c'_{enter}$ = 5+3=8. As $8 \not< \sigma$, the query has to go to the next level based on B^+-tree search for 10. In this level $[8, 22)$ is the smallest interval that covers $[10, 19]$. Now the total number of objects for the period is $C_{nl}(11).c'_{total}$ + $C_{nl}(22).c'_{enter}$ =3+1=4. As $4 < \sigma$ is true, the location $L1$ is not a DL during $[10, 19]$. As a result, it does not need to access the next level and do further processing for $L1$. Now consider another query $DLQ[6, 12, \theta]$ and let $\sigma = 3$. During the processing of this query on the tree for $L1$ shown in Fig. 3b, the pruning with *obs 1* does not work and the query will access the leaf level. However, the query does not have to fully compute the total number of objects during $[6, 12]$ as the sum up to time point 9 is: 2+1+1=4 (Fig. 3bb) and $4 > \sigma$. So, it can be deduced that the location is a DL during $[6, 12]$ without further processing.

V. Experimental Study

Experimental Setup: We implemented the mapping for CPS and pre-computed the aggregate information from the mapping records using C# and the DLT-Index using C. For all SQL queries, we use a leading RDBMS. The experiments were conducted on a laptop with an Intel Core i7 2.7 GHz processor with 8 GB main memory. The operating system is Windows 7 64 bit.

We use real RFID based baggage tracking data from the transfer system of terminal-3 of Copenhagen Airport (CPH). It has 11 CPS locations with 11 RFID readers deployed. After filtering some erroneous records there are 2.1 M tracking records for 220 K distinct bags collected during Dec 21, 2011 to Dec 02, 2013. As we do not have the exact values, the parameters ENLDs and EXLDs are generated. As most of the locations contain loops, after converting 2.1 M tracking records there are 787K Mapping records produced.

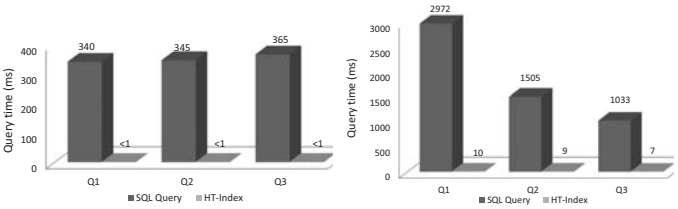
DLT-Index: The page size was set to 4KB and each entry of the DLT-tree was 20 bytes. This yields 204 entries per node.We use the above mapping table produced from the airport baggage tracking data and scaled it to 10M mapping records. For scaling, Max(ObjId) is added with existing ObjId for uniqueness and random time between 50 to 120 seconds are added with both $time_{start}$ and $time_{end}$ for each record. The semi-real data contains 3M distinct objects and total 17.3 M distinct time points distributed to the 11 symbolic locations.

Tree Construction: Before the tree construction the aggregate information C for each time point is precomputed. Then the trees are constructed for the time points including the aggregate information. Fig. 4a shows the effect of number of mapping records on the full tree construction time along with the aggregate information generation time. It shows that the tree construction time increases linearly with the increase in number of mapping records. Generating the aggregate information takes good amount of time as it has to access each record for finding distinct time points, sort and union them and compute the value of c_{total}, c_{enter} and c_{exit} at each time point.

Query Processing: Three random *DLQs* are generated for semi-real data and are processed with both pruning (P) and without pruning (NP). The node access for each of them is reported in Fig. 4b and 4c. In all cases, pruning accesses much fewer nodes compared to without pruning. In Fig. 4b the query time interval is kept fixed and the density threshold(θ) is changed. It shows that, for without pruning, a query has to access same number of nodes regardless of change in θ as it always has to compute total number of objects for full interval. However, for pruning, the number of node access decreases with increase in θ as the number of dense location decreases and σ increases (effect of pruning with *obs 1*) and at some point it becomes constant. In Fig. 4c, θ is kept fixed and the time interval length is changed. It seems that for without pruning, node access increases with increase in interval length as the query has to cover more time points to complete the interval. However, for pruning, node access decreases and become constant at some point. Here, any of the observation (*obs 1* or *obs 2*) can be the reason depending on the number

of dense locations. The query is effected by *obs 1* if number of dense location decreases and by *obs 2* if number of dense location increases. We also process three random queries that finds number of objects at each location for one month large interval with a given condition on the aggregate results (like group by with having condition in SQL) to see the effect of pruning on larger interval in the DLT-Index. Fig. 4d shows that the pruning technique access significantly less number of nodes compared to without pruning.

We randomly generate 3 point queries and also 3 interval queries that find number of objects at each location for a given time interval. The interval queries are for one year large interval. The queries are processed by both the DLT-Index and the relational DBMS. At first we have processed the queries inside the DBMS without creating any indices in the *MappingTable*. Then we have created both clustered index and non-clustered index on different columns and on combination of columns and processed the queries. However, the queries were fastest without any indices. So, we have reported the query processing durations without an index. For the fair comparison, the first query processing time for each query in the SQL is ignored for warm up and to load the data into main memory. Then we execute each query five times and take the average query time. Fig. 5a and 5b shows that the point query processing in DLT-Index is more than 340 times faster and some cases the interval query processing time is around 300 times faster.



(a) Point query (b) Interval group. query

Fig. 5: DLT-Index vs. RDBMS

VI. Related Work

Several papers address density queries and hot route queries on road networks [4], [7] in outdoor spaces. There are spatio-temporal data access methods for point and interval queries on temporal dimension in outdoor settings [9], [11] and in indoor settings [6]. These indexing techniques are capable to efficiently access individual records for a range of locations as well as for a time period. However, for a DLQ, aggregate information for each location has to be accessed. Our DLT-Index itself contains aggregate information and avoids accessing individual data records. There are also works for indexing spatio-temporal data for aggregate queries [8]. The aRB-tree [8] stores the aggregate information in the tree nodes. However, it counts same objects multiple times if the objects remain in the same location in several timestamps during the query interval. This problem is solved by an approximate approach in [10]. Unlike to the traditional range and point queries where the query generally does not access all the locations, the DLQ has to access all the locations' information to determine whether each location is a DL or not. So like the aRB-tree, we also maintain a separate tree for each location except the R-Tree part. The distance functions in the symbolic space are very far from the Euclidean distance, so the MBRs cannot approximate the distances well. Thus, the MBRs used in aRB-tree is not applicable in our case. As mentioned earlier our temporal indexing structure is motivated by [3], but we extend it to a symbolic spatio-temporal space and we additionally maintain necessary aggregate information in nodes to facilitate counting distinct objects for a time interval. The aggregate information stored in the non-leaf nodes helps achieve effective pruning in processing DLQs as well as interval and group by with having conditions.

VII. Conclusion

We proposed an approach to extract the dense locations from indoor tracking data. We developed a graph-based models for mapping the tracking records with the semantic location so that it is possible to know the entry and exit times of an object at a symbolic location. Then we proposed an indexing technique, the Dense Location Time Index (DLT-Index), that indexes aggregate information with the time points. We also proposed efficient techniques for index construction and query processing and pruning, for dense location queries on the DLT-Index. Our experimental evaluation on large amounts of real data shows that the DLT-Index can process queries efficiently and much faster than an RDBMS. The DLT-Index is also useful for general time interval indexing for efficient processing of queries like the number of distinct records for a specified time point or time interval.

Acknowledgment

This work is supported by the BagTrack project funded by the Danish National Advanced Technology Foundation under grant no. 010-2011-1.

References

[1] T. Ahmed, T. B. Pedersen, and H. Lu. Capturing hotspots for constrained indoor movement. In *SIGSPATIAL/GIS*, pages 462–465, 2013.

[2] T. Ahmed, T. B. Pedersen, and H. Lu. A data warehouse solution for analyzing rfid-based baggage tracking data. In *MDM (1)*, pages 283–292, 2013.

[3] R. Elmasri, G. T. J. Wuu, and Y.-J. Kim. The time index: An access structure for temporal data. In *VLDB*, pages 1–12, 1990.

[4] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective density queries on continuouslymoving objects. In *ICDE*, page 71, 2006.

[5] C. S. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *MDM*, pages 122–131, 2009.

[6] C. S. Jensen, H. Lu, and B. Yang. Indexing the trajectories of moving objects in symbolic indoor space. In *SSTD*, pages 208–227, 2009.

[7] X. Li, J. Han, J.-G. Lee, and H. Gonzalez. Traffic density-based discovery of hot routes in road networks. In *SSTD*, pages 441–459, 2007.

[8] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang. Indexing spatio-temporal data warehouses. In *ICDE*, pages 166–175, 2002.

[9] M. Romero, N. R. Brisaboa, and M. A. Rodríguez. The smo-index: a succinct moving object structure for timestamp and interval queries. In *SIGSPATIAL/GIS*, pages 498–501, 2012.

[10] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-temporal aggregation using sketches. In *ICDE*, pages 214–225, 2004.

[11] Y. Tao and D. Papadias. Mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In *VLDB*, pages 431–440, 2001.